XML in relationalen Datenbanken

Jost Enderle

Die zunehmende Bedeutung von XML zwingt auch die Hersteller relationaler Datenbanksysteme zur Bereitstellung entsprechender Schnittstellen. Einerseits sollte es möglich sein, relationale Daten im XML-Format auszugeben; andererseits ist es auch von Vorteil, wenn XML-Daten direkt im Datenbanksystem gespeichert werden können. Der vorliegende Artikel beschreibt verschiedene Ansätze und gibt Einblicke in aktuelle Implementierungen.

The increasing importance of XML also induces the vendors of relational database systems to provide adequate interfaces. On the one hand, it should be possible to retrieve relational data in XML format; on the other hand, it is also advantageous if XML data can be directly saved in the database. The present article describes several approaches and gives insights into current implementations.

Categories and subject descriptors:
H.2.3 [Database Management]: Languages – SQL; H.2.4 [Database Management]: Systems – relational databases, textual databases; I.7.2 [Document and Text Processing]: Document Preparation – XML

XML – Relationale Datenbanken – Textdatenbanken – SQL

1 Einleitung

Relationale Datenbanksysteme haben sich im Laufe der vergangenen zwanzig Jahre zur beherrschenden Technologie für die Speicherung und Verwaltung von Daten entwickelt. Dieser Erfolg beruht nicht zuletzt auf der herstellerunabhängigen Anfragesprache SQL, die die Abfrage und Manipulation der in den Datenbanktabellen gespeicherten Daten erlaubt. SQL wird bereits seit 1986 von amerikanischen (ANSI) und internationalen (ISO) Standardisierungsgremien normiert und liegt nun in der aktuellen Version SQL:1999 vor. Mit dieser Version wurden der Sprache eine Reihe von objektorientierten Eigenschaften wie benutzerdefinierte Typen und Methoden, Vererbung und Objektidentität hinzugefügt. Systeme, die solche Merkmale implementieren, werden oft als „objekt-relational“ bezeichnet.

Eine Erfolgsstory ganz anderer Art feierte in den letzten fünf Jahren die Auszeichnungssprache XML. Als Nachfolgerin von HTML entwickelt sie sich immer mehr zum Standard für die Darstellung von Inhalten im Internet. Ihre Erweiterbarkeit und Plattformunabhängigkeit prädestinieren sie als Sprache für den Datenaustausch in beliebigen Anwendungsgebieten. XML wurde vom W3C (World Wide Web Consortium) entwickelt und liegt seit 1998 in der Version 1.0 vor (als Empfehlung des W3C). Obwohl noch jung an Jahren, hat sie sich – ähnlich wie die Programmiersprache Java – im Zuge des Internet-Booms bereits zu einer etablierten Technologie entwickelt.

Wegen ihrer immensen Bedeutung und Verbreitung treffen relationale Datenbanksysteme und XML in der Praxis oft aufeinander. Riesige Mengen von Unternehmensdaten sind in relationalen Datenbanken gespeichert und müssen zur Darstellung und Übermittlung im Internet ins XML-Format umgewandelt werden. Andererseits gibt es schon große Mengen von XML-basierten Dokumenten, die nun in relationalen Datenbanken gespeichert und verwaltet werden sollen. Obwohl schon etliche Spezialösungen zur Speicherung von XML-Daten existieren, wie z. B. Dokumentmanagementsy-

Jost Enderle
Universität Ulm, Abt. Datenbanken und Informationssysteme, D-89069 Ulm
E-Mail: jost.enderle@informatik.uni-ulm.de

steme und spezielle XML-Datenbanken, kommt deren Einsatz oft nicht in Betracht, da die in XML darzustellenden Daten weiterhin in relationalen Systemen verwaltet werden oder schon bestehende XML-Daten zusammen mit anderen (relationalen) Daten verwendet und abgefragt werden sollen.

In Abschnitt 2 wird zunächst eine kurze Einführung zu XML gegeben. Anschließend werden in den Abschnitten 3 und 4 allgemeine Konzepte zur Speicherung und Ausgabe von XML-Daten in relationalen Systemen betrachtet. Abschnitt 5 behandelt den SQL/XML-Ansatz des SQL-Komitees der ISO. Lösungen verschiedener Hersteller werden dann in Abschnitt 6 näher beleuchtet.

2 XML – eine Einführung

XML (eXtensible Markup Language) [8] ist eine vereinfachte Untermenge von SGML (Standard Generalized Markup Language), einem textbasierten Datenformat für strukturierte Dokumente. Sie ist wie HTML (HyperText Markup Language) eine Auszeichnungssprache, die die Kennzeichnung einzelner Dokumentbestandteile mit sog. Tags erlaubt. Während HTML jedoch nur einen festen Satz von Tags besitzt (die im Wesentlichen der Formatierung von Text dienen), lassen sich in XML vom Benutzer neue Tags definieren. Diese werden dann im Allgemeinen nicht zur Formatierung eingesetzt (dies geschieht in XML separat durch sog. Stylesheets), sondern dienen der Anreicherung des Textes mit Metainformationen.

Beispiel:

```
<Leute>
  <Person Sprache = "deutsch">
    <Name>Alex</Name>
    <Alter>54</Alter>
    <Mail>alex@xml.de</Mail>
  </Person>
  <Person Sprache = "englisch">
    <Name>Berta</Name>
    <Alter>45</Alter>
    <Mail>berta@xml.com</Mail>
  </Person>
</Leute>
```

Dieses XML-Fragment enthält einige Informationen (Name, Alter, Mail-Adresse, Muttersprache) über zwei Personen (Alex und Berta). Wie man sieht, gibt es zu jedem Tag (. . .) auch ein End-Tag (. . .) Der von einem Tag und einem zugehörigen End-Tag eingeschlossene Inhalt wird als Element bezeichnet. Elemente können (eingeschachtelte) Subelemente besitzen; es ergibt sich somit eine hierarchische Dokumentstruktur. Neben Elementen gibt es auch Attribute (hier: „Sprache"), die direkt in ein (Start-) Tag integriert werden. Die Entscheidung, ob Informationen als Elemente oder Attribute modelliert werden sollen, ist nicht immer eindeutig („Sprache" hätte hier auch als Element modelliert werden können). Während Elemente auch Subelemente besitzen können, sind Attribute stets atomar.

XML ist gleichzeitig auch eine Metasprache, mit deren Hilfe sich andere Auszeichnungssprachen für spezielle Anwendungszwecke definieren lassen. Von dieser Möglichkeit wird in der Praxis auch reger Gebrauch gemacht, wie die große Zahl der mit Hilfe von XML definierten Sprachen beweist:

- CML (Chemical Markup Language)
- MathML (Mathematical Markup Language)
- BSML (Bioinformatic Sequence Markup Language)
- AIML (Astronomical Instrument Markup Language)
- cXML (CommerceXML)
- XFDL (Extensible Forms Description Language)
- XUL (User Interface Language)
- BML (Bean Markup Language)
- CPML (Call Policy Markup Language)
- . . .

Die Struktur solcher Sprachen lässt sich mit Hilfe von DTDs (Document Type Definitions) festlegen, die die Menge der zulässigen Tags definieren und festlegen, wie diese auf den Text angewandt werden können.

Beispiel: Eine mögliche DTD für das obige XML-Fragment

```
<!DOCTYPE Leute [
    <!ELEMENT Leute (Person*)>
    <!ELEMENT Person (Name, Alter,Mail)>
    <!ATTLIST Person Sprache CDATA
    #REQUIRED>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Alter (#PCDATA)>
    <!ELEMENT Mail (#PCDATA)>
]>
```

Wichtige W3C-Spezifikationen rund um XML

Tabelle 1

XML Path Language [22]	Sprache zur Adressierung der Inhalte (Teile) von XML-Dokumenten
XML Query [23, 24]	Abfragesprache für XML-Daten
XML Schema [25, 26, 27]	Sprache zur Spezifikation von XML-Dokumenttypen (ersetzt DTDs)
XSL Transformations [28]	Sprache zur Transformation von XML-Dokumenten in XML-Dokumente mit anderer Struktur und/oder anderem Vokabular; basiert auf XPath

Hinter dem Schlüsselwort DOCTYPE muss das Wurzelelement des Dokuments angegeben werden (in diesem Fall „Leute"). Der Stern (*) ist eine Kardinalitätsangabe und bedeutet „keinmal oder beliebig oft". Das Element „Leute" enthält also beliebig viele (oder keine) Subelemente des Typs „Person". Weitere mögliche Kardinalitätsangaben sind „ + " (einmal oder beliebig oft) und „?" (keinmal oder einmal). „Person" besitzt wiederum die drei Subelemente „Name", „Alter" und „Mail", und das genau in dieser Reihenfolge. „Person" besitzt ebenfalls ein Attribut „Sprache", dessen Typ „CDATA" (Character DATA, d. h. beliebiger Text) ist und auf jeden Fall angegeben werden muss (#REQUIRED; Alternative wäre #IMPLIED, d. h. optional). „Name", „Alter" und „Mail" sind jeweils vom Typ „PCDATA" (Parsed Character DATA, d. h. Text, der keine Tags mehr enthält). In DTDs gibt es keine verschiedenen Datentypen für (Blatt-)Elemente, alles ist Text.

Vor kurzem wurde vom W3C ein Nachfolgestandard für DTDs namens „XML Schema" definiert [25, 26, 27] (Tabelle 1). Wesentliche Kennzeichen von XML Schema (gegenüber DTDs) sind die XML-Syntax (d. h. ein XML-Schema ist auch ein XML-Dokument) und die Einführung von Datentypen (auch benutzerdefinierte). Auf XML Schema wird an dieser Stelle nicht näher eingegangen.

3 Speicherung von XML-Daten in relationalen Datenbanken

Viele Hersteller relationaler Datenbanken unterstützen heute schon die Speicherung von XML-Daten in ihren Systemen. Im Allgemeinen werden dabei zwei verschiedene Ansätze verfolgt. Beim ersten Ansatz erfolgt die Speicherung des gesamten XML-Dokuments als CLOB (Character Large Object) in einer Tabellenspalte. Dieser Ansatz lässt sich zwar einfach realisieren, bietet jedoch nur eingeschränkte Funktionalität bzw. Effizienz, wenn auf einzelne Teile der XML-Daten zugegriffen werden soll. Da das Datenbanksystem nicht über die innere Struktur des XML-Dokuments Bescheid weiß, müssen entsprechende Funktionen bereitgestellt werden, die einen Zugriff auf einzelne Teile des Dokuments ermöglichen.

Beim zweiten Ansatz werden die zu speichernden XML-Dokumente zunächst strukturell analysiert und dann die im XML-Dokument enthaltenen Daten in relationaler Form in einer oder mehreren Tabellen abgespeichert. Dabei werden z. B. Blattelemente des XML-Dokuments auf entsprechende Spalten abgebildet und hierarchische Strukturen auf entsprechende Tabellen, die über Primär- und Fremdschlüssel miteinander verknüpft werden. Dazu muss eine feste Abbildung zwischen dem Schema der zu speichernden Dokumente und dem entsprechenden Relationenschema definiert sein. Dieser Ansatz ist weitaus mächtiger als der erste, da nun die komplette Funktionalität des Datenbanksystems (Anfrageverarbeitung, Indexe usw.) zur Verwaltung der eingespeicherten Daten bereitsteht.

Prinzipiell ist also der zweite Ansatz dem ersten vorzuziehen. Allerdings erweist sich dessen Realisierung in der Praxis als problematisch. Da sich das hierarchische Datenmodell von XML wesentlich vom Relationenmodell unterscheidet, ist es manchmal sehr schwer bis unmöglich, eine vernünftige

Abbildung des Schemas der einzulesenden XML-Daten auf ein Relationenschema zu finden.

Hersteller relationaler Datenbanksysteme handeln pragmatisch und führen eine Klassifizierung der zu speichernden XML-Dokumente ein. Dokumente, die eine hinreichend kompatible Struktur zum Relationenmodell aufweisen, werden als „strukturiert“ oder „datenorientiert“ bezeichnet; für diese lässt sich meist auf einfache Weise ein entsprechendes Relationenschema definieren. Wo sich die Abbildung allerdings als sehr schwierig erweist, wird häufig von „unstrukturierten“ oder „dokumentenorientierten“ XML-Daten gesprochen. Hier bleibt dann nur die Speicherung als CLOB übrig.

Diese Aufteilung der XML-Dokumente in zwei Klassen mag auf den ersten Blick etwas willkürlich erscheinen, erweist sich in der Praxis aber als recht brauchbar. Im Folgenden werden die beiden Ansätze näher beleuchtet.

3.1 Speicherung von XML-Daten als CLOB

Wie oben besprochen, lässt sich das Schema mancher XML-Dokumente nur schwer auf ein entsprechendes Relationenschema abbilden. Es folgt eine Aufzählung der möglichen Ursachen.

XML-Daten weisen eine *hierarchische Schachtelung* auf, relationale Daten jedoch nicht. Es muss eine entsprechende Abbildung auf mehrere Tabellen gefunden werden. Außerdem lassen sich XML-Schemas definieren (es wird hier nicht zwischen DTDs und XML-Schemas i. e. S. unterschieden), die eine rekursive Schachtelung von Elementen erlauben. Eine Abbildung auf ein Relationenschema ist dann prinzipiell nicht mehr möglich.

Während die Komponenten eines XML-Dokuments aufgrund der Serialisierung eine *feste Reihenfolge* aufweisen, sind die Tupel einer Tabelle ungeordnet. Bei der simplen Abbildung von XML-Elementen auf Tabellentupel geht also die Ordnungsinformation verloren. Soll die Reihenfolge erhalten bleiben, müssen zusätzliche Spalten zur Speicherung dieser Information eingeführt werden.

XML-Elemente können *gemischte Inhalte* aufweisen. Darunter versteht man die gleichwertige Aneinanderreihung von Text und Subelementen innerhalb eines übergeordneten XML-Elements. Typisches Beispiel hierfür sind Formatierungsanweisungen:

```
<Absatz>
  Normaler Text<fett>Fetter Text
  </fett>Normaler Text
</Absatz>
```

Das entsprechende DTD-Fragment dazu könnte folgendermaßen aussehen:

```
<!ELEMENT Absatz (#PCDATA|fett)*>
<!ELEMENT fett (#PCDATA)>
```

Der Operator „|“ kennzeichnet eine Alternative. Das heißt, ein Absatz kann Text oder das Subelement „fett“ enthalten, und das beliebig oft hintereinander (Operator „*“). Damit entsteht ein gemischter Inhalt.

Es ist unklar, wie gemischte Inhalte auf Tabellenspalten abgebildet werden sollen. Eine direkte Abbildung von XML-Elementen auf Tabellenspalten ist hier sicherlich nicht sinnvoll.

XML lässt bei der Modellierung semantisch identischer Sachverhalte *syntaktisch unterschiedliche Lösungen* zu. Zum Beispiel können zur Beschreibung der Eigenschaften eines XML-Elements sowohl XML-Attribute als auch Subelemente verwendet werden:

```
<Angestellter Gehalt = "80000"/>
```

ist z. B. äquivalent zu

```
<Angestellter ><Gehalt>80000
</Gehalt ></Angestellter>
```

Wird das Angestelltenelement auf eine Angestelltentabelle mit einer Spalte „Gehalt“ abgebildet, muss neben der eigentlichen Information auch gespeichert werden, ob beim Auslesen der XML-Daten aus der Datenbank XML-Elemente oder -Attribute erstellt werden sollen.

XML-Dokumente enthalten neben den eigentlichen Daten Konstrukte, die *Metadaten oder zusätzliche Funktionalität* bereitstellen:

- Kommentare,
- Verarbeitungsanweisungen für Programme, die auf das Dokument zugreifen,
- sog. CDATA-Blöcke, die die Interpretation von Zeichen als Markup verhindern (Escape-Mechanismus),

- sog. Entities zur Definition von Textmakros,
- Angabe des XML-Schemas, dem das Dokument entspricht.

Es ist ebenfalls unklar, wie solche Konstrukte auf Tabellen abgebildet werden sollen.

Nach dieser Aufzählung dürfte klar sein, dass es nicht möglich bzw. sinnvoll ist, XML-Dokumente beliebig komplexer Struktur auf ein Relationenschema abzubilden, ohne dabei einen gewissen Informationsverlust in Kauf zu nehmen. Für Dokumente, die nach dem Auslesen aus der Datenbank den exakt gleichen Aufbau wie vor dem Einspeichern haben sollen, bietet sich daher nur die Speicherung als CLOB an. Dokumente, auf die dieses Kriterium zutrifft, dienen meist der Visualisierung der in ihnen gespeicherten Daten. Hier spielen die Erhaltung der Elementordnung und gemischte Inhalte eine große Rolle. Bei einem XML-Dokument, das z. B. in einem Web-Browser angezeigt werden soll, dürfen keine Komponenten vertauscht werden; ansonsten erscheint der in ihm enthaltene Text nicht mehr in der richtigen Reihenfolge.

Relationale Systeme, die eine Speicherung von XML-Dokumenten als CLOB erlauben, bieten oft entsprechende Suchfunktionen an, die die Lokalisierung einzelner Teile des Dokuments erlauben. Textindexe können dabei die Suche beschleunigen. Neben der primitiven Textsuche werden manchmal auch XML-spezifische Suchmöglichkeiten geboten; z. B. können über XPath-Ausdrücke [22] bestimmte Knoten im Dokument lokalisiert werden.

Sollen ausschließlich Dokumente dieser Art gespeichert und verwaltet werden, bietet sich die Verwendung eines speziellen Dokument- oder Contentmanagementsystems an.

3.2 Abbildung der XML-Daten auf ein Relationenschema

Nicht in jedem Fall muss mit XML-Dokumenten beliebig komplexen Inhalts gerechnet werden. Wo XML hauptsächlich als Medium zum Datenaustausch zwischen verschiedenen Systemen eingesetzt wird (z. B. Übermittlung von Bestellungen, Produktdaten etc.), hat man es meist mit sehr strukturierten Daten zu tun, die sich auch einfach auf ein Relationenschema abbilden lassen. Gemischte Inhalte treten hier praktisch nicht auf, und auch die Reihenfolge von Geschwisterelementen (z. B. verschiedene Angestellte einer Abteilung) spielt hier in der Regel keine Rolle. Hierarchische Strukturen ohne Rekursivität lassen sich unmittelbar auf mehrere Tabellen mit entsprechenden Fremdschlüsselbeziehungen abbilden.

Dieser Ansatz bietet entscheidende Vorteile gegenüber dem ersten. Da sich die eingelesenen XML-Daten hier nicht mehr von anderen relationalen Daten unterscheiden, können diese mit den normalen Mechanismen des Datenbanksystems bearbeitet werden (SQL-Anfragen, Indexe etc.). Die Wiederherstellung des XML-Dokuments ist allerdings aufwändiger, da nun die Daten aus verschiedenen Tabellen zusammengesucht werden müssen (und nicht einfach aus dem CLOB ausgelesen werden können).

Bezüglich der konkreten Abbildung zwischen XML- und Relationenschemas existieren schon einige Untersuchungen [4, 9, 13, 18], die auch Richtlinien und Algorithmen zur (automatischen) Schematransformation enthalten. Die verschiedenen Verfahren schränken die Klasse der transformierbaren XML-Schemas mehr oder weniger ein (z. B. keine gemischten Inhalte) und zeigen auch „Notlösungen“ für problematische Fälle auf (z. B. Erhalten der Ordnung, Unterscheidung zwischen Elementen und Attributen). Bourret [4] schlägt z. B. folgendes Verfahren zur Erstellung eines relationalen Schemas aus einer DTD vor:

1. Erzeuge für jeden Elementtyp mit Subelementen bzw. gemischtem Inhalt eine Tabelle mit einer Primärschlüsselspalte.
2. Erzeuge für jeden Elementtyp mit gemischtem Inhalt eine separate Tabelle zur Speicherung der PCDATA-Komponente. Diese Tabelle wird mit der Tabelle des Elementtyps über deren Primärschlüssel verknüpft.
3. Erzeuge in der Tabelle eines Elementtyps für jedes einwertige[1] Attribut des Elementtyps und jeden nur einmal vorkommenden Subelementtyp, der nur PCDATA enthält, eine Spalte. Wenn der Subelementtyp bzw. das Attribut optional sind, muss die Spalte den Wert NULL annehmen können.
4. Erzeuge für jedes mehrwertige Attribut eines Elementtyps und für jeden Subelementtyp, der ein Mehrfachvorkommen erlaubt, jedoch nur PCDATA enthält, eine separate Tabelle zur Speicherung der Werte. Diese Tabelle wird mit der Tabelle des Elementtyps über deren Primärschlüssel verknüpft.

[1] Ein Attribut kann auch eine Liste von Werten enthalten.

5. Verknüpfe für jeden Subelementtyp mit Subelementen oder gemischtem Inhalt die Tabelle des Elementtyps mit der Tabelle des Subelements über den Primärschlüssel des Elementtyps.

3.3 Hybride Ansätze

In der Praxis erweisen sich Mischformen der beiden oben erläuterten Ansätze oft als nützlich. Soll auf XML-Dokumente in der Datenbank nur lesend zugegriffen werden, bietet es sich z. B. an, die Dokumente sowohl als CLOB in der Datenbank abzulegen als auch eine entsprechende (verlustbehaftete) Abbildung auf ein Relationenschema durchzuführen. Bei diesem Vorgehen können die gespeicherten Dokumente einerseits exakt rekonstruiert werden, andererseits lassen sich effiziente Anfragen auf den in Tabellen gespeicherten Daten stellen. Manche Systeme lassen es sogar zu, Updates auf dem im CLOB gespeicherten XML-Dokument direkt auf die entsprechenden relationalen Daten zu propagieren.

Manchmal ist es auch hilfreich, ein XML-Dokument auf mehrere CLOBs abzubilden. Soll z. B. ein Buch im XML-Format in der Datenbank gespeichert werden, kann es sinnvoll sein, einzelne Abschnitte des Dokuments in verschiedenen CLOBs zu speichern. Die darüber liegende Struktur des XML-Dokuments wird dagegen auf Tabellen abgebildet. Damit sind direkte Anfragen über die Struktur (z. B. welche Abschnitte gehören zu welchem Kapitel) und Metadaten (z. B. Autor) des Dokuments möglich. Innerhalb der eigentlichen Textabschnitte kann mit entsprechenden Textfunktionen gearbeitet werden.

4 Ausgabe von relationalen Daten im XML-Format

Wie oben dargestellt, können bei der Abbildung von XML-Schemas in Relationenschemas einige Probleme auftreten. Solange bestehende relationale Daten einfach im XML-Format ausgegeben werden sollen, lassen sich schnell praktikable Lösungen finden. Die Ergebnisstruktur einer SQL-Anfrage kann mit entsprechenden Rowset-, Row- und Column-Tags nachgebildet werden, wie z. B. von Bourret [4] und Martin et al. [13] beschrieben:

```
<ROWSET>
  <ROW>
    <COLUMN1>...</COLUMN1>
    <COLUMN2>...</COLUMN2>
      ...
    <COLUMNn>...</COLUMNn>
  </ROW>
  <ROW>
    <COLUMN1>...</COLUMN1>
    <COLUMN2>...</COLUMN2>
      ...
    <COLUMNn>...</COLUMNn>
  </ROW>
    ...
</ROWSET>
```

Entspricht dieses von der Datenbank gelieferte (kanonische) Format nicht den gewünschten Vorstellungen, bietet es sich an, XSL-Transformationen [28] auf die ausgegebenen XML-Daten anzuwenden. Mit deren Hilfe ist es möglich, XML-Dokumente über Baumtransformationen in fast beliebiger Weise umzustrukturieren (z. B. Umwandlung eines Subelements in ein Attribut).

Sollen von der Datenbank prinzipiell komplexer strukturierte XML-Daten (die nicht dem kanonischen Tabellenformat entsprechen) ausgegeben werden, erscheint es sinnvoll, XML-orientierte Anfragesprachen zu verwenden. Shanmugasundaram et al. [18] schlagen vor, XML-QL-Anfragen [7] des Benutzers in entsprechende SQL-Anfragen umzuwandeln und die zurückgelieferten Ergebnisse entsprechend der XML-QL-Anfrage zu strukturieren. Andere Ansätze propagieren die Erweiterung von SQL um strukturgebende Operatoren, mit deren Hilfe komplexe XML-Ergebnisse direkt erzeugt werden können. Ein solches Verfahren wird im nächsten Abschnitt vorgestellt.

5 Der Ansatz des SQL-Komitees: SQL/XML

Im vorigen Jahr erkannte das SQL-Standardisierungskomitee der ISO die Notwendigkeit, sich um eine Normung der Schnittstelle zwischen SQL und XML zu kümmern, um einen Wildwuchs proprietärer Lösungen zu vermeiden. Da das für die Standardisierung von XML (und verwandten Technologien) zuständige W3C bis zu diesem Zeitpunkt keine besonderen Maßnahmen in dieser Richtung unternommen hatte, wurde von Melton [14] die Erweiterung von SQL:1999 um einen neuen Teil SQL/XML vorgeschlagen, der sich ausschließlich mit der Beziehung zwischen SQL und XML auseinander setzt. Als mögliche Inhalte wurden u. a. festgelegt:

Die aktuellen Teilstandards von SQL (SQL:1999 umfasst nur die ersten fünf Teile)

Tabelle 2

ISO 9075–1 SQL/Framework	Übersicht für den kompletten Standard
ISO 9075–2 SQL/Foundation	Kerndokument, beschreibt alle grundlegenden SQL-Features
ISO 9075–3 SQL/CLI (Call-Level Interface)	Aufruf von SQL-Befehlen durch Programmiersprachen
ISO 9075–4 SQL/PSM (Persistent Stored Modules)	Erweiterung von SQL um imperative Sprachkonstrukte; gespeicherte Prozeduren und Funktionen
ISO 9075–5 SQL/Bindings (Host Language Bindings)	Einbettung von SQL-Befehlen in Programmiersprachen
ISO 9075–7 SQL/Temporal	Unterstützung von zeitbasierten Datentypen
ISO 9075–9 SQL/MED (Management of External Data)	Unterstützung von externen (nicht im Datenbanksystem gespeicherten) Daten
ISO 9075–10 SQL/OLB (Object Language Bindings)	Beschreibung von SQLJ (in Java eingebettetes SQL)
ISO 9075–11 SQL/Schemata	Beschreibung von SQL-Schemas
ISO 9075–13 SQL/JRT (SQL Routines and Types Using the Java Programming Language)	Unterstützung von Java-Routinen und -Typen in SQL
ISO 9075–14 SQL/XML	Unterstützung von XML in SQL

- Spezifikationen für die Darstellung von SQL-Daten (d. h. Tupeln, Tabellen, Views, Anfrageergebnissen) im XML-Format,
- Spezifikationen bezüglich der Abbildung zwischen SQL-Schemas und XML-Schemas,
- Spezifikation der Art und Weise, wie SQL mit XML-Daten umgehen kann.

Die Arbeiten zu diesem neuen Teilstandard (ISO 9075–14; Tabelle 2) stecken noch in den Anfängen. Bisher wurden nur die Abbildungen der SQL-Zeichensätze auf die XML-Zeichensätze, der SQL-Bezeichner auf die XML-Namen und der vordefinierten SQL-Datentypen auf die entsprechenden XML-Schema-Datentypen genauer spezifiziert [21]. Für die Darstellung von SQL-Daten im XML-Format liegen schon konkrete Vorschläge vor [12, 11, 19], von denen einer hier kurz vorgestellt werden soll.

Kulkarni et al. [12] schlagen einige Erweiterungen zu SQL vor, die die Konstruktion beliebig komplexer (auch rekursiver) XML-Dokumente direkt in SQL erlauben. Anhand des folgenden Beispiels werden diese Erweiterungen näher betrachtet.

Gegeben sei das folgende Relationenschema:

```
Abteilungen(ID INTEGER, AbtName VAR-
CHAR(20))
Angestellte(ID INTEGER, AbtID INTE-
GER, AngName VARCHAR(20))
```

Es gibt Abteilungen und Angestellte. Beide Tabellen haben ein ID-Feld, das jeweils als Schlüssel dient. Jeder Angestellte referenziert über AbtID die Abteilung, zu der er gehört. Eine Abteilung soll nach folgendem Muster in XML ausgegeben werden:

```
<Abteilung id = "abt1">
  <AbtName>Forschung</AbtName>
  <Angestellte>
    <Angestellter id = "ang1">
      <AngName>Maier</AngName>
    </Angestellter>
    <Angestellter id = "ang2">
      <AngName>Bauer</AngName>
    </Angestellter>
    ...
  </Angestellte>
</Abteilung>
```

Die folgende Anfrage liefert das gewünschte Ergebnis:

```
SELECT abt.AbtName, Abt-
Func(abt.ID,abt.AbtName,AngList
(abt.ID))
FROM Abteilungen abt
```

Die Anfrage erzeugt für jede Abteilung in der Abteilungstabelle ein Tupel mit zwei Spalten. Die erste Spalte enthält den Namen der Abteilung und die zweite Spalte die XML-Darstellung der Abteilung. AbtFunc und AngList sind skalare SQL-Funktionen. AngList erzeugt die XML-Darstellung aller Angestellten einer Abteilung, und AbtFunc erzeugt die Darstellung für die Abteilung selbst. AbtFunc ist folgendermaßen definiert:

```
CREATE FUNCTION AbtFunc(id INTEGER,
                        abtname VARCHAR(20),
                        anglist CLOB(10000))
  RETURNS CLOB(10000) LANGUAGE XML
  RETURN
    <Abteilung id = {id}>
      <AbtName>{abtname}</AbtName>
      <Angestellte>{anglist}
      </Angestellte>
    </Abteilung>
```

Der XML-Rückgabewert dieser Funktion wird über ein XML-Template definiert. Im Template wird {x} durch den Wert des Parameters x ersetzt. Man beachte, dass XML-Fragmente hier in Variablen des Typs CLOB gespeichert werden. Die Definition für die AngList-Funktion sieht wie folgt aus:

```
CREATE FUNCTION AngList(abtid INTE-
GER)
  RETURNS CLOB(10000) LANGUAGE SQL
  RETURN
    SELECT XMLAGG('<Angestellter
                   id = '||ang.ID||'>'||
                    ang.AngName||
                   '</Angestellter>')
    FROM Angestellte ang
    WHERE ang.AbtID = abtid
```

Die AngList-Funktion erzeugt die Liste von Angestellten, die zu einer Abteilung gehören, im XML-Format. Die neue Aggregatfunktion XMLAGG erzeugt aus der ihr übergebenen Zeichenkette das Ergebnis-CLOB.

6 Was machen die Hersteller?

Im Folgenden werden die Ansätze von Oracle und IBM DB2 bezüglich der Speicherung und Ausgabe von XML-Daten näher beleuchtet.

6.1 Oracle9i

Banerjee et al. [2] beschreiben die in der aktuellen Oracle-Version [15] vorhandene XML-Funktionalität.

6.1.1 Der CLOB-Ansatz

Oracle stellt seit der Version 9i einen speziellen Datentyp „XMLType“ bereit, dessen Instanzen intern als CLOB gespeichert werden. XMLType kann wie jeder andere Oracle-Datentyp zur Definition von Tabellenspalten oder innerhalb von gespeicherten Prozeduren und benutzerdefinierten Funktionen zur Definition von Variablen, Parametern und Rückgabewerten verwendet werden.

Auf XMLType sind systemseitig bereits einige nützliche Funktionen definiert, die die Erzeugung von XMLType-Instanzen und das Auslesen von Fragmenten aus solchen Instanzen erlauben:

- createXML() nimmt einen Wert vom Typ VARCHAR oder CLOB, überprüft, ob es sich um ein gültiges XML-Fragment handelt, und liefert eine entsprechende XMLType-Instanz zurück.
- extract() liefert für eine XMLType-Instanz und einen XPath-Ausdruck ein oder mehrere XML-Fragmente (vom Typ XMLType), die dem XPath-Ausdruck entsprechen.
- existsNode() prüft für eine XMLType-Instanz und einen XPath-Ausdruck, ob in der Instanz ein dem Ausdruck entsprechendes XML-Fragment enthalten ist.

Mit Hilfe von createXML() lassen sich z. B. direkt XML-Fragmente in XMLType-Spalten einfügen:

```
INSERT INTO Tabelle_mit_XMLType_
Spalte
VALUES (..., sys.XMLType.createXML
('<Tag1>...XML_Text...</Tag1>'))
```

Die folgende Anfrage liefert für einen gegebenen XPath-Audruck (hier: /Auftrag/Kunde) die qualifi-

zierenden XML-Fragmente (also<Kunde>-Fragmente innerhalb von<Auftrag>-Elementen) zurück:

```
SELECT t.XML_Spalte.extract
('/Auftrag/Kunde')
FROM Tabelle_mit_XMLType_Spalte t
WHERE ...
```

Enthalten die selektierten XML-Fragmente nur noch reinen Text (d. h. keine weiteren XML-Subelemente mehr), kann dieser auch direkt angefragt werden (ohne umgebende Tags):

```
SELECT ...
FROM Tabelle_mit_XMLType_Spalte t
WHERE t.XML_Spalte.extract
      ('/Auftrag/Kunde/Name/text()'
      ).getStringVal() LIKE '%Maier%'
```

Zu weiteren Anfragemöglichkeiten auf XML-Dokumenten mit Hilfe von XPath sei auf [22] verwiesen.

Die InterMedia-Text-Cartridge (spezielles Oracle-Erweiterungspaket, das die Textsuche in CLOBs erlaubt) wurde so erweitert, dass nun auch eine indexunterstützte Suche auf XML-Daten möglich ist. Mit Hilfe eines speziellen CONTAINS-Operators kann nach Text innerhalb bestimmter XML-Elemente oder -Attribute gesucht werden:

```
SELECT *
FROM Tabelle_mit_XMLType_Spalte
WHERE CONTAINS(XMLType_Spalte,
'Tulpenweg WITHIN Adresse')>0
```

Diese Anfrage liefert alle Tupel der Tabelle_mit_XMLType_Spalte zurück, die in der XMLType_Spalte ein XML-Dokument enthalten, das an beliebiger Stelle den folgenden Text enthält:

```
<Adresse>... Tulpenweg ...</Adresse>
```

Durch Verschachtelung von WITHIN-Operatoren kann die Suche mit CONTAINS auf größere Zusammenhänge ausgedehnt werden:

```
... CONTAINS(Spalte,'(Irrweg WITHIN
Strasse) WITHIN Adresse')>0
```

Der CONTAINS-Operator unterstützt auch die Textsuche mit Hilfe von XPath-Ausdrücken. Auf die genaue Syntax wird hier nicht näher eingegangen.

Ändernde Zugriffe auf einzelne Komponenten einer XMLType-Instanz sind (wie beim zugrunde liegenden Typ CLOB) in Oracle leider nicht möglich. XMLType-Instanzen können nur als Ganzes überschrieben werden.

6.1.2 Ausgabe von Anfrageergebnissen im XML-Format

Oracle bietet die Möglichkeit, innerhalb von SQL-Anfragen XML-Fragmente zu generieren. Dazu werden zwei neue SQL-Funktionen bereitgestellt:

- SYS_XMLGEN nimmt ein einzelnes Argument eines eingebauten oder benutzerdefinierten Typs und wandelt es (je nach Typ) in ein entsprechendes XML-Element um.
- SYS_XMLAGG nimmt eine Menge von XML-Fragmenten und hängt diese aneinander (wird im Folgenden nicht mehr betrachtet).

Handelt es sich beim Argument von SYS_XMLGEN um einen skalaren Wert, werden einfach entsprechende Tags für das Argument erzeugt. Die Anfrage

```
SELECT SYS_XMLGEN(Gehalt)
FROM Angestellte
WHERE PersNr = 1234
```

könnte z. B. die folgende Ausgabe liefern:

```
<?xml version = '1.0'?>
<Gehalt>6000</Gehalt>
```

Bei der Umwandlung werden auch sämtliche objekt-relationalen Typen von Oracle unterstützt. Handelt es sich beim Argument von SYS_XMLGEN um einen Objekttyp mit mehreren Attributen, werden für diese im XML-Resultat separate Subelemente erzeugt. Oracle-Kollektionen (Arrays und verschachtelte Tabellen) werden entsprechend auf Listen von Elementen abgebildet.

Oracle stellt dem Benutzer auch ein Programm namens XSU (XML SQL Utility) zur Verfügung, das die Ergebnisse ganzer SQL-Anfragen ins XML-Format umwandelt. Diese Abbildung ist relativ statisch und liefert für rein relationale Anfragen die schon in Abschnitt 4 beschriebene XML-Struktur mit Row-

set-, Row- und Column-Tags. Wird z. B. eine Anfrage wie

```
SELECT PersNr, Name, Gehalt
FROM Angestellte
```

als Textstring an XSU übergeben, so wird ein Resultat ähnlich dem folgenden zurückgeliefert:

```
<?xml version = '1.0'?>
<ROWSET>
  <ROW num = "1">
    <PersNr>12345</PersNr>
    <Name>Huber</Name>
    <Gehalt>80000</Gehalt>
  </ROW>
  <ROW num = "2">
    ...
  </ROW>
  ...
</ROWSET>
```

Durch Setzen bestimmter Parameter lassen sich die Schlüsselwörter ROWSET und ROW in der XML-Ausgabe durch konkrete Namen ersetzen (im obigen Beispiel z. B. ROWSET durch „Angestelltenliste" und ROW durch „Angestellter"). Außerdem kann bei der Formulierung der Anfrage festgelegt werden, welche Werte auf XML-Attribute (und nicht auf Elemente) abgebildet werden sollen. Wie oben beschrieben, werden auch hier bei der XML-Ausgabe sämtliche objekt-relationalen Typen von Oracle unterstützt.

Es kann vorkommen, dass die von XSU aus den Anfrageergebnissen automatisch erzeugten XML-Dokumente nicht die von der Anwendung benötigte Struktur haben. Da der eigentliche Abbildungsprozess nur wenige Freiheitsgrade bietet, muss entweder die Eingabe für XSU so angepasst werden, dass nach der Transformation das Gewünschte herauskommt, oder das Ergebnis muss nachträglich noch entsprechend adaptiert werden:

- Mit Hilfe sog. Objektsichten ist es möglich, relationalen Tabellen eine komplexe (objekt-relationale) Struktur aufzuprägen, die dann von XSU in eine entsprechend verschachtelte XML-Struktur abgebildet wird.
- Der XML-Parser von Oracle stellt einen XSL-Prozessor zur Verfügung, der die automatische Transformation von XML-Anfrageergebnissen erlaubt.

6.1.3 Einlesen von XML-Daten in Tabellen

XSU erlaubt auch das Einlesen von XML-Daten in Tabellen. Dazu müssen an XSU als Parameter der Name einer bestehenden Tabelle und ein XML-Dokument übergeben werden, dessen Schema mit dem Schema der spezifizierten Tabelle korrespondiert (nach obigem Muster). XSU erzeugt dann entsprechende SQL-INSERT-Befehle, die die Daten des XML-Dokuments in die Tabelle einspeichern. Passen das XML- und das relationale Schema nicht (exakt) überein, kann man sich wieder mit Objektsichten und XSL-Transformationen behelfen.

6.2 IBM DB2 Universal Database, Version 7

Cheng und Xu [5] beschreiben die Funktionalität des in der aktuellen DB2-Version bereitgestellten XML-Extenders [10].

6.2.1 Der CLOB-Ansatz

DB2 stellt für die Speicherung von XML-Dokumenten in einer Spalte drei spezielle Datentypen zur Verfügung:

- XMLCLOB: Speicherung des XML-Dokuments als CLOB in DB2 (außerhalb der Tabelle);
- XMLVARCHAR: Speicherung des XML-Dokuments als VARCHAR in DB2 (innerhalb der Tabelle);
- XMLFile: Speicherung des XML-Dokuments als externe Datei.

Beim Einspeichern der XML-Dokumente in eine Tabellenspalte können diese gegenüber einer DTD validiert werden, die der Spalte zugeordnet ist. Damit kann gewährleistet werden, dass nur Dokumente eines bestimmten Schemas in der Spalte gespeichert werden.

Neben der reinen Speicherung der XML-Dokumente als Textobjekte erlaubt DB2 die Extraktion der Werte einzelner Elemente und Attribute aus den Dokumenten, die dann in Spalten so genannter „Seitentabellen" abgespeichert werden. Auf diese Daten kann dann wie auf andere relationale Daten zugegriffen werden (nur lesend); insbesondere lassen sich die Spalten der Seitentabellen indexieren. Bei Änderungen des XML-Dokuments werden auch die Seitentabellen automatisch aktualisiert. Auf diese Weise kann auf häufig benötigte Daten in den XML-Dokumenten sehr effizient zugegriffen werden.

Die Abbildung von Elementen und Attributen der zu speichernden XML-Dokumente auf entsprechende Seitentabellen wird in sog. DAD-Dateien (DAD = Document Access Definition) spezifiziert. Jede Spalte einer Seitentabelle wird darin einem bestimmten Element oder Attribut der DTD zugeordnet (über einen XPath-Ausdruck), die den zu speichernden XML-Dokumenten zugrunde liegt. Auf die genaue Struktur der DAD-Dateien und das Schema der Seitentabellen wird hier nicht näher eingegangen.

Beim Anlegen einer Tabelle, die eine XML-Spalte enthält, wird dieser Spalte eine DAD-Datei zugeordnet. Diese wird vom System ausgelesen, und die entsprechenden Seitentabellen werden automatisch erstellt. Ebenso werden beim Einspeichern von XML-Dokumenten in die XML-Spalte die Seitentabellen automatisch aktualisiert.

Zur Speicherung der Dokumente in der XML-Spalte stellt DB2 verschiedene Funktionen zur Typumwandlung bereit. Es sind z. B. folgende INSERT-Befehle möglich:

```
INSERT INTO Tabelle_mit_XMLVAR-
CHAR_Spalte
VALUES (..., db2xml.XMLVARCHAR(:Var-
charVariable_mit_XML))

INSERT INTO Tabelle_mit_
XMLCLOB_Spalte
VALUES (..., XMLCLOBFromFile('c:\da-
tei.xml'))
```

Auf einzelne Elemente und Attribute der XML-Dokumente kann direkt zugegriffen werden. Im einfachsten Fall werden die gesuchten Daten bereits in Seitentabellen verwaltet. Ein Zugriff auf die eigentlichen XML-Dokumente kann dann unter Umständen völlig vermieden werden. Aber auch wenn die gesuchten Informationen nur in den XML-Dokumenten selbst vorliegen (z. B. weil überhaupt keine Seitentabellen angelegt wurden), ist ein einfacher Zugriff möglich. DB2 stellt (ähnlich wie Oracle) spezielle „Extraktionsfunktionen" bereit, die anhand einer XPath-Angabe jedes beliebige Element oder Attribut eines XML-Dokuments auslesen können:

```
SELECT extractINTEGER(XML_Spalte,
'/Auftrag/Kunde/KundenNr')
FROM Tabelle_mit_XML_Spalte
WHERE ...
```

Für jeden DB2-Datentyp gibt es eine separate Extraktionsfunktion. Der aus dem XML-Dokument ausgelesene Knoten wird in den entsprechenden DB2-Datentyp umgewandelt (soweit möglich). Die XML_Spalte kann einen beliebigen XML-Typ (XMLVARCHAR, XMLCLOB oder XMLFile) aufweisen.

Genauso einfach lassen sich Updates auf einzelnen Elementen und Attributen durchführen:

```
UPDATE Tabelle_mit_XML_Spalte
SET XML_Spalte = Update(XML_Spalte,
'/Autrag/Kunde/Name','Maier')
WHERE ...
```

DB2 stellt auch ausgefeilte Textsuchfunktionen für XML-Dokumente bereit. Dabei wird zwischen struktureller Textsuche (Suche nur in bestimmten Elementen oder Attributen, spezifiziert durch XPath-Ausdrücke) und Volltextsuche (Suche im gesamten Dokument) unterschieden. Auf die Syntax des entsprechenden CONTAINS-Operators wird hier nicht näher eingegangen.

6.2.2 Abbildungen zwischen DTDs und Relationenschemas

DB2 gestattet die Speicherung von XML-Daten in relationaler Form. Die dafür nötige Abbildung zwischen dem Schema der zu speichernden XML-Dokumente und dem Schema der entsprechenden Tabellen kann vom Benutzer frei gewählt werden. Je nachdem, ob schon bestehende XML-Dokumente auf Tabellen abgebildet oder schon bestehende relationale Daten im XML-Format ausgegeben werden sollen, wird man entweder zu einem XML-Schema ein passendes Relationenschema oder zu einem Relationenschema ein passendes XML-Schema suchen. Die exakte Abbildung zwischen den Elementen und Attributen der XML-Dokumente und den Spalten der Tabellen wird wieder in einer DAD-Datei festgelegt.

DB2 stellt zwei Funktionen zur Verfügung, die die Generierung bzw. Zerlegung eines XML-Dokuments entsprechend einer DAD-Spezifikation vornehmen:

- dxxGenXML() erzeugt ein XML-Dokument aus einer Menge von Tabellen;

- dxxShredXML() zerlegt ein XML-Dokument in eine Menge von Tabellen.

Die Spezifikation einer entsprechenden DAD-Datei ist recht aufwändig und wird hier nicht beschrieben. Auch auf die genaue Benutzung der beiden Abbildungsfunktionen wird nicht näher eingegangen.

Bei der Bearbeitung der im relationalen Format vorliegenden XML-Daten muss stets im Auge behalten werden, welche Auswirkungen Änderungen der Daten auf die evtl. später wieder zu erzeugenden XML-Dokumente haben. So kann z. B. die Löschung eines Tupels in einer Tabelle die Löschung eines kompletten XML-Dokuments nach sich ziehen, wenn das Tupel dem Top-Level-Element des XML-Dokuments entspricht.

7 Fazit

Wieder einmal scheinen es relationale Datenbanksysteme zu schaffen, sich den wechselnden Anforderungen des Datenbankmarktes anzupassen. Wurden bisher zur Verwaltung semistrukturierter Daten wie XML aus Effizienzgründen meist spezielle Systeme verwendet, können heute in den meisten Fällen auch vorhandene relationale Systeme zur Speicherung dieser Daten eingesetzt werden. Die Hersteller sind eifrig dabei, die gebotene XML-Funktionalität weiter auszubauen, und das SQL-Komitee sorgt durch entsprechende Standards für langfristige Kompatibilität zwischen den verschiedenen Implementierungen. Nur wo besondere Anforderungen an die Verwaltung von XML-Dokumenten bestehen, wie z. B. Versionierung oder lange Transaktionen, sollte weiterhin auf entsprechende Spezialsysteme (z. B. Contentmanagementsysteme) zurückgegriffen werden.

Literatur

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Hove: Morgan Kaufmann 2000
2. Banerjee, S., Krishnamurthy, V., Krishnaprasad, M., Murthy, R.: Oracle8 i – The XML Enabled Data Management System. Proc. 16th ICDE, San Diego 2000
3. Beech, D.: A Snapshot of XSQL: A Query Language for XML. Discussion paper, WG3:HEL-029, H2–2000–440, August 2000
4. Bourret, R.: XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm (2001)
5. Cheng, J., Xu, J.: XML and DB2. Proc. 16th ICDE, San Diego 2000
6. Deckers, M., Eisele, R., Petkovic, D.: German National Position about the Direction of SQL/XML. Discussion Paper, WG3:HEL-053, September 2000
7. Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: XML-QL: A Query Language for XML. Submission to the W3C, August 1998. http://www.w3.org/TR/NOTE-xml-ql/
8. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 6 October 2000. http://www.w3.org/TR/REC-xml
9. Florescu, D., Kossmann, D.: Storing and Querying XML Data using an RDBMS. Bulletin of the Technical Committee on Data Engineering 22(3), 27–34 (1999)
10. IBM DB2 Universal Database – XML Extender: Administration and Programming, Version 7. IBM Corporation, 2000
11. Krishnamurthy, V., Krishnaprasad, M.: Effectively Publishing XML Data Using Object-Relational Technology. Discussion paper, WG3:HEL-032, H2–2000–457
12. Kulkarni, K., Pirahesh, H., Shanmugasundaram, J., Shekita, E., Yannakopoulos, A., Zeidenstein, K.: SQL Extensions for Publishing Relational Data as XML Documents. Discussion paper, WG3:HEL-032, H2–2000–449R1
13. Martin, D., Birbeck, M., Kay, M. et al.: Professional XML. Chicago: Wrox Press 2000
14. Melton, J.: XML-Related Specifications (SQL/XML). Subproject proposal, WG3: HEL-026R2, H2–2000–331R2, October 2000
15. Oracle9 i Application Developer's Guide – XML, Release 1 (9.0.1). Oracle Corporation, June 2001. Part No. A88894–01
16. Quin, L.: Open Source XML Database Toolkit: Resources and Techniques for Improved Development. New York: John Wiley & Sons 2000
17. Shanmugasundaram, J., Shekita, E., Barr, R., Carey, M., Lindsay, B., Pirahesh, H., Reinwald, B.:Efficiently Publishing Relational Data as XML Documents. Discussion paper, WG3:HEL-033, H2–2000–458, August 2000 (published in: Proc. 26th VLDB Conf., Cairo 2000)
18. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. 25th VLDB Conf., Edinburgh 1999, pp. 302–314
19. Shaw, P.: A JDBC-Oriented XML DTD for SQL Result Sets. Discussion paper, WG3: HEL-035, H2–2000–186, March 2000
20. Zeidenstein, K., Kulkarni, K., Pirahesh, H., Shanmugasundaram, J., Shekita, E.: Discussion on SQL/XML Program of Work. Discussion paper, WG3:HEL-031, H2–2000–448, August 2000
21. XML-Related Specifications (SQL/XML). ISO-ANSI Working Draft, WG3:YYJ-012, H2–2001–149, June 2001
22. XML Path Language (XPath), Version 1.0. W3C Recommendation, 16 November 1999. http://www.w3.org/TR/xpath
23. XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, 7 June 2001. http://www.w3.org/TR/query-datamodel/
24. XML Query Requirements. W3C Working Draft, 15 February 2001. http://www.w3.org/TR/xmlquery-req/
25. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-0/
26. XML Schema Part 1: Structures. W3C Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-1/
27. XML Schema Part 2: Datatypes. W3C Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-2/
28. XSL Transformations (XSLT), Version 1.0. W3C Recommendation, 16 November 1999. http://www.w3.org/TR/xslt